Counting in Trees along Multidirectional Regular Paths*

Everardo Bárcenas
INRIA Rhône Alpes
Everardo.Barcenas-Patino@inria.fr

Pierre Genevès
CNRS
Pierre.Geneves@inria.fr

Nabil Layaïda
INRIA Rhône Alpes
Nabil.Layaida@inria.fr

ABSTRACT

We propose a tree logic capable of expressing simple cardinality constraints on the number of nodes selected by an arbitrarily deep regular path with backward navigation. Specifically, a sublogic of the alternation-free μ–calculus with converse for finite trees is extended with a counting operator in order to reason on the cardinality of node sets. Also, we developed a bottom-up tableau-based satisfiability-checking algorithm, which resulted to have the same complexity than the logic without the counting operator: a simple exponential in the size of a formula.

This result can be seen as an extension of the so-called graded-modalities introduced in [18], which allows counting constraints only on immediate successors, with conditions on the number of nodes accessible by an arbitrary recursive and multidirectional path. This work generalizes the optimal complexity bound: $2^{O(n)}$ where n is the length of the formula, shown in [11], for satisfiability of the logic extended with such counting constraints.

Finally, we identify a decidable XPath fragment featuring cardinality constraints on paths with upward/downward recursive navigation, in the presence of XML types.

1. INTRODUCTION

The μ–calculus (MC) [17] is a logic that comes from the application of modal and temporal logics to program verification. The two main features of MC are: its great expressive power, it subsumes many of the logics used in systems verification [4]; and its low computational complexity [26]. In [24], it is argued the finite tree model property is the responsible of the relatively easy evaluation of MC. Also, due to the finite tree model property, MC became a powerful tool to reason on tree structures [7]. Converse modalities were added to MC in [25] (MCC), achieving a powerful balance between expressivity and succinctness, unfortunately the finite model property was lost, increasing the difficulties in the implementation of decision procedures for MCC. A μ–calculus with converse with the finite tree model property was introduced in [11] with $2^{O(n)}$ complexity, where n corresponds to the length of a formula. Furthermore, [11] presents an efficient implementation of the decision procedure along with an application to XPath decision problems in the presence of XML types.

The necessity to reason on counting issues in transition systems quickly led to the first attempts to extend some modal logics to reason with counting constraints [10]. Following these earliest attempts, [13] introduced a limited form of counting in description logics, where the occurrence number of nodes can be imposed only on contiguous neighbors of a certain node. The consideration of transitive roles increases the expressive power of counting in description logics but leads to undecidability [14]. In order to be able to post numerical constraints on nodes reacheables by recursive and multidirectional paths in tree structures, we formulate an alternation-free MCC extended with a counting operator where the models are finite tree structures (MCCC). Also, we introduce a satisfiability algorithm for MCCC whose complexity is simple exponential in the size of a formula.

In the context of efficient type checking for XML-based programming languages where XML types and XPath queries are used as first class language constructs, XPath decision problems in the presence of XML types, such as DTDs and XML Schemas are very important. The emptiness test of XPath expressions and XPath containment, are the core decision problems due to their importance in issues as optimization of expressions [12], control flow analysis of XSLT [19], checking integrity constraints [8], checking access control in XML security applications [9], among others.

It is worth noticing that one of the main difficulty in XPath decision problems is the consideration of a possibly infinite quantification over a set of trees. Among the features also affecting the difficulty of such problems we find the presence of XML types [3, 11], the combination of downward and up-down navigation with recursion in trees [25, 3, 11], comparison of data values of infinite domain [3, 15], and cardinality constraints on node sets [6, 22]. It is already known that the consideration of the whole set, as well as some subsets, of such features leads to undecidability [21, 3]. Numerical con-

*Copyright is held by the author/owner. International Workshop on Programming Language Techniques for XML (PLAN-X 2009), January 24, 2009, Savannah, Georgia.

straints on paths w.r.t. a constant can be possed hardcoding ordering but clearly leading to an exponential blow-up. In this work we identify a decidable fragment of XPath featuring cardinality constraints and upward/downward navigation with recursion in the presence of XML types. In order to solve XPath decision problems, we translate both XPath and XML type expressions into MCCC formulas.

Regular tree type expressions subsume most XML types in use today [20]. When control over the number of occurrences is needed in XML types, expressions like T^+ are used for denoting at least one occurrence of expression T and possibly arbitrarily many of them. XML Schema introduce a more fine-grained control with the attributes `minoccur` and `maxoccur`. Such attributes allow expressing that a type expression T occurs for at least n times and/or less that m times. Graded modalities [18] are useful for avoiding exponential blowups that would otherwise occur if this kind of constraints are translated naively into their regular representation.

In a more general setting from the perspective of counting constraints, XPath expressions like $\rho_1[\rho_2 \leq n]$ are commonly used. Such an expression selects the set of nodes in a tree that can be reached by means of the path ρ_1, and additionally, the number of nodes reached from there by means of the path ρ_2 is less or equal than the natural number n. Since we are considering upward/downward navigation with recursion, the paths ρ_1 and ρ_2 can denote nodes in any part of the tree. Therefore, we need the ability to count in any part of the tree structure, possibly in the presence of tree type constraints.

2. RELATED WORK

The simpler attempt to capture navigation on trees is by means of first order logic [2]. The use of second order logic, in particular, monadic second order logic with two succesors served as a much more expressive tool to reason about trees [2]. A variant of propositional dynamic logic was proposed in [1] to study trees, achieving an efficient decision procedure but with limited expressivity. The propositional μ–calculus [17] turned out to be a very useful alternative to reason about tree structures [7]. In order to allow backward navigation in the models of μ–calculus, converse modalities have been also considered in [25]. Unfortunately, the finite model property was lost. It has been syntactically restored in [23] and [11] with $2^{O(n \log n)}$ and $2^{O(n)}$ complexities, resp., where n corresponds to the length of a formula. In addition, [11] describes an efficient implementation of the decision procedure along with an application to XPath decision problems in the presence of XML types. All these approaches do not consider counting constraints.

Besides the already mentioned limited form of counting in transition systems introduced for modal logics in [10], and further developed for description logics in [13], graded modalities have been also considered for the μ–calculus in [18]. Two similar, and more sophisticated approaches to counting, have been considered in [6, 21]. A modal logic, called sheaves logic, is introduced in [6] to count through paths. The consideration of variables in both arguments of a binary cardinality operator gives significant additional expressivity to this approach, but on the other hand, counting constraints are still restricted to children nodes. In [21], additionally, recursive navigation is allowed by means of a fixpoint first order logic, but still the numerical constraints are only permitted on children nodes. Proper automata theory is developed to prove decidability of the resulting logics in both approaches.

In order to balance expressivity, succinctness and complexity, we choose to consider backward navigation and recursion in cardinality constraints, but we restrict the presence of variables only on one argument of the binary cardinality operator. It is worth remembering the conjunction of forward/backward navigation with recursion in cardinality constraints and the presence of variables in both arguments of a binary counting operator leads to undecidability [16].

3. OUTLINE

We first present our logic in Section 4, then in Section 5, we introduce a XPath fragment followed by a translation of it into the logic. We proceed to present a correct satisfiability algorithm for the logic in Section 6. Finally, we present our conclusions and draw directions for some further research in Section 7.

4. THE LOGIC

This section presents a modal logic to reason on counting issues on finite tree structures. This logic is an extension of the alternation-free μ–calculus with converse first introduced in [11]. The extension consists in the consideration of a counting operator $\leq$ that allows a new kind of formulas: $\phi \leq n$, where ϕ is a formula and n is a natural number. The interpretation of $\phi \leq n$ is either the set of all tree nodes if there is n or less than n nodes that satisfy ϕ in the tree, otherwise it is the empty set. Consider for example the formula $\phi_1 \wedge (\phi_2 \leq n)$, named ψ. Since ϕ_1 and ϕ_2 are any kind of formula, and recursive forward/backward navigation in the tree is allowed, then ψ can denote the set of nodes that satisfy ϕ_1, provided the set of their ancestors, or descendants, or any other set of nodes in the tree denoted by ϕ_2 has a cardinality which is equal or less than n.

Some syntactic restrictions are considered in order for the least and greatest fixpoint operators to coincide, and thus to keep the logic closed under negation, in the manner of [11]. The restrictions are named cycle-freeness of the formulas and they are presented just after the syntax and semantics of the logic.

4.1 Syntax and Semantics

We consider a countable set of *propositions, variables.* The set of *modalities* is defined as $\{1, 2, \bar{1}, \bar{2}\}$, where for any modality m we have $\bar{\bar{m}} = m$.

Modalities are used for modeling basic navigation in trees: $\langle 1 \rangle$ navigates from a node to its first child, while $\langle 2 \rangle$ navigates from a node to its first sibling. Converse programs allow for symmetric backward navigation. It is common knowledge that binary trees represent unranked (n-ary) trees without loss of generality. This "first-child" and "next-sibling" encoding is also used in [11].

We inductively define the set of *formulas* as:

- x, p and $\top$ are formulas if p is a proposition and x is a variable ($\top$ is the true formula), and

- $\neg\phi, \phi_1 \vee \phi_2, \phi_1 \wedge \phi_2, \langle m \rangle \phi$, let $\overline{x_i.\phi_i}$ in $\phi, \phi \leq n$ are also formulas when ϕ, ϕ_1, ϕ_2 and $\overline{\phi_i}$ are formulas, m is a modality, $\overline{x_i}$ are variables and n is a natural number.

Since regular tree types are often mutually recursively defined, we consider the n–ary version of the fixpoint operator, making the translation of regular expression types more succinct.

It will be written $\mu x.\phi$ instead of let $x.\phi$ in ϕ, $\perp$ instead of $\neg\top$, $\phi > n$ instead of $\neg(\phi \leq n)$, and $\phi = n$ instead of $(\phi \leq n) \wedge (\phi > n - 1)$.

The set of *subformulas* of a formula ϕ is defined as usual, and it is denoted F_ϕ.

A *tree structure* is a tuple (N, R, L) where:

- N is a finite set named the *nodes*;

- R is a function from a binary relation, between the nodes and the modalities, to the nodes, such that:
  - $R(n, m) \neq n$ for any n and m;
  - $R(n_1, m) = n_2$ iff $R(n_2, \bar{m}) = n_1$ for any n_1, n_2 and m;
  - there is a exactly one node r, named the *root*, such that both $R(r, \bar{1})$ and $R(r, \bar{2})$ are not defined; and
  - exactly one function $R(n, \bar{1})$ or $R(n, \bar{2})$ is defined for all the nodes except for the root; and

- L is a function from the nodes to the propositions.

As a very simple example consider the tuple $S = (N, R, L)$ such that

- $N = \{n_0, n_1, n_2\}$,

- $R(n_0, 1) = n_1$, $R(n_1, 2) = n_2$,

- $L(n_0) = p_1$, $L(n_1) = p_2$ and $L(n_2) = p_1$,

which clearly satisfies the requirements to be considered as a tree structure where:

- n_0 is the root and it is labelled with p_1; and

- n_1 and n_2 are the children of n_0, and they are labelled with p_2 and p_1, resp.

The *semantics of formulas* is defined w.r.t. a tree structure (N, R, L) and a variable interpretation V (a binary relation between variables and nodes) as:

$$
\begin{array}{lcl}
[\![\top]\!]^S_V & = & N \\
[\![p]\!]^S_V & = & \{n \mid L(n) = p\} \\
[\![x]\!]^S_V & = & \{n \mid (n, x) \in V\} \\
[\![\neg\phi]\!]^S_V & = & N \setminus [\![\phi]\!]^S_V \\
[\![\phi_1 \wedge \phi_2]\!]^S_V & = & [\![\phi_1]\!]^S_V \cap [\![\phi_2]\!]^S_V \\
[\![\phi_1 \vee \phi_2]\!]^S_V & = & [\![\phi_1]\!]^S_V \cup [\![\phi_2]\!]^S_V \\
[\![\langle m \rangle \phi]\!]^S_V & = & \{n \mid R(n, m) \in [\![\phi]\!]^S_V\} \\
[\![\phi \leq n]\!]^S_V & = & \begin{cases} N & \text{if } |[\![\phi]\!]^S_V| \leq n \\ \emptyset & \text{otherwise} \end{cases} \\
[\![\text{let } \overline{x_i.\phi_i} \text{ in } \phi]\!]^S_V & = & [\![\phi]\!]^S_{V[\overline{N_i/x_i}]}
\end{array}
$$

where $V[M/x]$ means $(n, x) \in V$ for all $n \in M$, and N_i is the least fixpoint of ϕ_i w.r.t. to x_i, i.e., $N_i = \bigcap\{M \mid [\![\phi_i]\!]^S_{V[M/x_i]} \subseteq M\}$.

We say two formulas ϕ_1 and ϕ_2 are *equivalent* iff $[\![\phi_1]\!]^S_V = [\![\phi_2]\!]^S_V$ for some S and V, and a formula ϕ is said to be *satisfiable* iff $[\![\phi]\!]^S_V \neq \emptyset$ for some S and V.

EXAMPLE 4.1. *Consider the following formulas:*

$$
\begin{array}{lcl}
D(\phi) & := & \langle 1 \rangle \mu x.\phi \vee \mu y.\langle 1 \rangle (x \vee y) \vee \langle 2 \rangle y \\
A(\phi) & := & \mu x.\langle \bar{1} \rangle (\phi \vee x) \vee \langle \bar{2} \rangle x
\end{array}
$$

Given a tree structure S, $A(\phi)$ denotes the set of ancestor nodes of the nodes denoted by ϕ in S, whereas $D(\phi)$ denotes the set of descendant nodes of ϕ. Now, consider

$$\phi_0 := a \wedge (A(b) \vee D(c))$$

ϕ_0 is satisfied for nodes in S which are named a, and either have at least one ancestor named b, or have at least one descendant named c. The formula

$$\phi_1 := \phi_0 > 5$$

is satisfied by all nodes of a tree in which there are more than 5 nodes verifying ϕ_0. Notice that in a tree where there are 5 or less nodes verifying ϕ_0, then ϕ_1 is not satisfied by any node. Finally, the formula

$$\phi_2 := \phi_1 \wedge \phi_0$$

is satisfied by nodes verifying ϕ_0 in a tree S if and only if there are more than 5 of them in S.

4.2 Cycle-Free Formulas

We now describe a syntactic restriction over formulas that ensure the logic is closed under negation [11]. This restriction forbids cycles when navigating in a tree, so that infinite testing of a node against a subformula is avoided. Intuitively, this restriction excludes formulas that use both a modality and its converse in front of a variable of the same fixpoint subformula.

The *unwinding* of a formula $\mu x.\phi$, written $exp^n(\mu x.\phi)$ for a natural number n, is inductively defined as:

- $exp^0(\mu x.\phi) = \mu x.\phi$, $exp^1(\mu x.\phi) = \phi[\mu x.\phi/x]$, and

- $exp^k(\mu x.\phi) = exp^{k-1}(\phi[exp^1(\mu x.\phi)/\mu x.\phi])$.

We will write $exp(\phi)$ instead of $exp^1(\phi)$.

The set of *unfoldings* of a formula ϕ is composed by the formulas $\phi[(exp^k(\phi_0)[\bot/\mu x.\phi_0])/\mu x.\phi_0]$ for each $\mu x.\phi_0 \in F_\phi$ and some k. We will refer to an unfolding of a formula ϕ as $unf(\phi)$.

Considering a formula ϕ, if $\mu x.\phi_0 \in F_\phi$, $\langle m\rangle\phi_1 \in F_{exp^k(\mu x.\phi_0)}$ for some k, $\langle\bar{m}\rangle\phi_2 \in F_{\phi_1}$, and $x \in F_{\phi_2}$, then we say ϕ *is not a cycle-free formula*. For cycle-free formulas, some easy consequences of Lemma 4.2 from [11] are:

- there is a equivalent unfolding for each formula;
- the logic is closed under negation and thus without lost of generality we can consider formulas in negation normal form: formulas where negation occurs only in front of propositions and formulas of the form $\langle m\rangle\top$ or $\phi \le n$;
- also w.l.o.g., we can consider only closed formulas (formulas where variables do not occur free) where, in addition, variables only occur under the scope of a modality.

To illustrate the concept of cycle-freeness consider the formula $\mu x.\langle 1\rangle(\phi \vee \langle\bar{1}\rangle x)$, which is not cycle-free, because the variable x, used for recursion, is under the scope of the modality 1 and its converse $\bar{1}$, creating a loop in the navigation when testing the safisfiability of the formula on a node.

5. XPATH

XPath [5] is a powerful query language for XML documents. Here, we present a large fragment of XPath covering major features of the XPath recommendation [5] including a form of counting constraints. Since XML documents are tree structures, it is natural to translate XPath expressions into formulas in order to analyze such expressions. Here, we present a translation of XPath into the logic presented in Section 4.

5.1 Syntax and Semantics

The set of XPath expressions is defined as follows:

$XPath \ni e$	$::=$			XPath expression
			$/\rho$	absolute path
		$\mid$	ρ	relative path
		$\mid$	$self::*[\mathrm{count}(\rho) \le n]$	cardinality constraint
		$\mid$	$self::*[\mathrm{count}(\rho) > n]$	cardinality constraint
		$\mid$	$e_1 \mid e_2$	union
		$\mid$	$e_1 \cap e_2$	intersection
$Path\ \ p$	$::=$			path
			ρ_1/ρ_2	path composition
		$\mid$	$\rho[q]$	qualified path
		$\mid$	$a::p$	step with node test
		$\mid$	$a::*$	step
$Qualif\ \ q$	$::=$			qualifier
			q_1 and q_2	conjunction
		$\mid$	q_1 or q_2	disjunction
		$\mid$	not q	negation
		$\mid$	ρ	path
$Axis\ \ a$	$::=$			tree navigation axis
			child | *self* | *parent*	
		$\mid$	*descendant* | *descself*	
		$\mid$	*ancestor* | *ancself*	
		$\mid$	*fsibling* | *psibling*	
		$\mid$	*following* | *preceding*	

where p is a proposition and n a natural number.

Consider the $XPath$ expression

```
/child::university/child::department/child::lab
```

which intuitively means the navigation from the root of a XML document through the nodes named `university` to its `department` child nodes and to its `lab` child nodes. We obtain from this evaluation all the `lab` nodes in the document which can be reached by such navigation. The consideration of other *axis* than *child* in $XPath$ expressions implies more sofisticated navigations, as for example the *axis* expression *ancestor*, which involve a backward recursive navigation. Additionally, it is possible to filter the selection of nodes using the boolean expressions between brackets named $qualifiers$, which can test the existence/absence of paths, and enforce cardinality constraints.

EXAMPLE 5.1. *Consider the expressions:*

$$e_0 \quad := self :: a[ancestor :: b \ or \ descendant :: c]$$
$$e_1 \quad := self :: *[count(e_0) > 5]$$

e_0 selects the nodes in a tree satisfying ϕ_0, introduced in Example 4.1. As for e_1, it selects the nodes satisfying ϕ_2, also introduced in Example 4.1.

Before defining the formal semantis of $XPath$ expressions we introduce some definitions.

We define the bijection $^\star : Axis \mapsto Axis$ as

$$\begin{array}{llllll} self^\star & = & self; & child^\star & = & parent; \\ fsibling^\star & = & psibling; & descendant^\star & = & ancestor; \\ descself^\star & = & ascself; & following^\star & = & preceding. \end{array}$$

such that if $a_1^\star = a_2$ then $a_2^\star = a_1$.

Given a tree structure $S = (N, R, L)$, we write $n_1 \xrightarrow{\widetilde{m}} n_{l+1}$, where $n_1, n_{l+1} \in N$ and $\widetilde{m}$ is a sequence of l modalities or an empty sequence. If $\widetilde{m}$ is not the empty sequence, then there are $n_2, n_3, \ldots, n_l \in N$ s.t. $R(n_i, m_i) = n_{i+1}$ $(i = 1, \ldots, l)$.

Now, we define the *semantics* of the XPath expressions w.r.t. a tree structure $S = (N, R, L)$ and a set $C \subseteq N$, called the context, as follows:

$$
\begin{array}{lcl}
\llbracket \rho \rrbracket_C^S & = & |\rho|_C^S \\
\llbracket /\rho \rrbracket_C^S & = & \llbracket \rho \rrbracket_{C \cup \{r\}}^S \\
\llbracket \mathit{self}{::}{*}[\mathrm{count}(\rho) \leq n] \rrbracket_C^S & = & \{t \mid t \in N \wedge |(\rho)_{\{t\}}^S| \leq n\} \\
\llbracket \mathit{self}{::}{*}[\mathrm{count}(\rho) \leq n] \rrbracket_C^S & = & \{t \mid t \in N \wedge |(\rho)_{\{t\}}^S| > n\} \\
\llbracket e_1 \mid e_2 \rrbracket_C^S & = & \llbracket e_1 \rrbracket_C^S \cup \llbracket e_2 \rrbracket_C^S \\
\llbracket e_1 \cap e_2 \rrbracket_C^S & = & \llbracket e_1 \rrbracket_C^S \cap \llbracket e_2 \rrbracket_C^S
\end{array}
$$

In the same context than we define the semantics of *Path* expressions as follows:

$$
\begin{array}{lcl}
(a :: \tau)_C^S & = & \llbracket a \rrbracket_C^S \cap \llbracket \tau \rrbracket^S \\
(\rho_1/\rho_2)_C^S & = & (\rho_2)_{(\rho_1)_C^S}^S \\
(\rho[q])_C^S & = & (\rho)_C^S \cap \|q\|_{(\rho)_C^S}^S
\end{array}
$$

The semantics of qualifiers differs from the paths in the sense than the paths select nodes, in contrast with the qualifiers which filter nodes, then we define:

$$
\begin{array}{lcl}
\|q_1 \text{ and } q_2\|_C^S & = & \|q_1\|_C^S \cap \|q_2\|_C^S; \\
\|q_1 \text{ or } q_2\|_C^S & = & \|q_1\|_C^S \cup \|q_2\|_C^S; \\
\|\text{not } q\|_C^S & = & N \setminus \|q\|_C^S; \\
\|a : \tau\|_C^S & = & \llbracket a^\star \rrbracket_{\llbracket \tau \rrbracket^S}^S; \\
\|\rho_1/\rho_2\|_C^S & = & \|\rho_1\|_{\|\rho_2\|_C^S}^S; \\
\|\rho[q]\|_C^S & = & \|\rho\|_{\|q\|_C^S}^S;
\end{array}
$$

where $\tau \in \{p, *\}$ and

$$
\begin{array}{lcl}
\llbracket p \rrbracket^S & = & \{n \mid L(n) = p\} \\
\llbracket * \rrbracket^S & = & N
\end{array}
$$

Finally, the *axis* are interpreted as follows:

$$
\begin{array}{lcl}
\llbracket \mathit{self} \rrbracket_C^S & = & C; \\
\llbracket \mathit{child} \rrbracket_C^S & = & \{n \mid c \xrightarrow{1,\tilde{2}} n, \forall c \in C\} \\
\llbracket \mathit{fsibling} \rrbracket_C^S & = & \{n \mid c \xrightarrow{\tilde{2}} n, \forall c \in C\} \\
\llbracket \mathit{psibling} \rrbracket_C^S & = & \{n \mid c \xrightarrow{\tilde{\bar{2}}} n, \forall c \in C\} \\
\llbracket \mathit{parent} \rrbracket_C^S & = & \{n \mid c \xrightarrow{\tilde{\bar{2}},\bar{1}} n, \forall c \in C\} \\
\llbracket \mathit{descendant} \rrbracket_C^S & = & \{n \mid c \xrightarrow{1,\widetilde{\bar{1},\tilde{2}}} n, \forall c \in C\} \\
\llbracket \mathit{descself} \rrbracket_C^S & = & \llbracket \mathit{self} \rrbracket_C^S \cup \llbracket \mathit{descendant} \rrbracket_C^S \\
\llbracket \mathit{ancestor} \rrbracket_C^S & = & \{n \mid c \xrightarrow{\widetilde{\bar{1},\tilde{\bar{2}}},\bar{1}} n, \forall c \in C\} \\
\llbracket \mathit{ancself} \rrbracket_C^S & = & \llbracket \mathit{self} \rrbracket_C^S \cup \llbracket \mathit{ancestor} \rrbracket_C^S \\
\llbracket \mathit{following} \rrbracket_C^S & = & \llbracket \mathit{descself} \rrbracket_{\llbracket \mathit{fpsibling} \rrbracket_{\llbracket \mathit{ancself} \rrbracket_C^S}^S}^S \\
\llbracket \mathit{preceding} \rrbracket|_C^S & = & \llbracket \mathit{descself} \rrbracket_{\llbracket \mathit{psibling} \rrbracket_{\llbracket \mathit{ancself} \rrbracket_C^S}^S}^S
\end{array}
$$

5.2 Translation

We now provide a translation of XPath expressions to logical formulas w.r.t a formula c named the context. The formula resulted from the translation of an XPath expression e w.r.t. a context c, will be written $F(e)_c$.

$$
\begin{array}{lcl}
F(/\rho)_c & = & F(\rho)_{r_c} \\
F(\rho)_c & = & F_1(\rho)_{c \wedge \circledS \wedge \circledS = 1} \\
F(\mathit{self} :: *[\mathrm{count}(\rho) \leq n])_c & = & F_1(\rho)_c \leq n \wedge F_2(\rho)_\top \\
F(\mathit{self} :: *[\mathrm{count}(\rho) > n])_c & = & F_1(\rho)_c > n \wedge F_2(\rho)_\top \\
F(e_1 \mid e_2)_c & = & F(e_1)_c \vee F(e_2)_c \\
F(e_1 \cap e_2)_c & = & F(e_1)_c \wedge F(e_2)_c \\
r_c & = & (\mu x. \neg\langle\bar{1}\rangle\top \vee \langle\bar{2}\rangle x) \wedge \\
 & & (\mu y. c \wedge \circledS \vee \langle 1\rangle y \vee \langle 2\rangle y)
\end{array}
$$

The translation of a relative path marks the initial context with $\circledS$. For absolute paths, the translation takes the formula r_c as the initial context. r_c navigates to the root.

In the translation of cardinality constraints, notice that $F_1(\rho)_c$ is duplicated. This is necessary to perform a sort of XPath-like "local" counting (as explained in Example 4.1).

Although $F_1(\rho)_c$ is duplicated in the translation, an important observation from a complexity point-of-view is that the size of the Lean (as defined in Section 6) does not increase with this duplication. As a result, the translation of an XPath expression remains linear in terms of the number of elements in the Lean.

$$
\begin{array}{lcl}
F_1(a :: p)_c & = & F(a)_c \wedge p \\
F_1(a :: *)_c & = & F(a)_c \\
F_1(\rho_1/\rho_2)_c & = & F_1(\rho_2)_{F_1(\rho_1)_c} \\
F_1(\rho[q])_c & = & F_1(\rho)_c \wedge F_2(q)_\top
\end{array}
$$

The translation of an XPath expression ρ_1/ρ_2 holds for all nodes accessed through ρ_2 from those nodes accessed through ρ_1. The translation of an expression like $\rho[q]$ represent the nodes accessed though ρ and from which q holds.

$$
\begin{array}{lcl}
F_2(q_1 \text{ and } q_1)_c & = & F_2(q_1)_c \wedge F_2(q_2)_c \\
F_2(q_1 \text{ or } q_1)_c & = & F_2(q_1)_c \vee F_2(q_2)_c \\
F_2(\text{not } q)_c & = & \neg F_2(q)_c \\
F_2(a :: p)_c & = & F(a^\star)_{p \wedge c} \\
F_2(a :: *)_c & = & F(a^\star)_{\top \wedge c} \\
F_2(\rho_1/\rho_2)_c & = & F_2(\rho_1)_{F_2(\rho_2)_c} \\
F_2(\rho[q])_c & = & F_2(\rho)_{F_2(q)_c} \\
F(self)_c & = & c \\
F(child)_c & = & \mu x.\langle \bar{1} \rangle c \vee \langle \bar{2} \rangle x \\
F(fsibling)_c & = & \mu x.\langle \bar{2} \rangle c \vee \langle \bar{2} \rangle x \\
F(psibling)_c & = & \mu x.\langle 2 \rangle c \vee \langle 2 \rangle x \\
F(parent)_c & = & \langle 1 \rangle \mu x.c \vee \langle 2 \rangle x \\
F(descendant)_c & = & \mu x.\langle \bar{1} \rangle (c \vee x) \vee \langle \bar{2} \rangle x \\
F(descself)_c & = & \mu x.c \vee (\mu y.\langle \bar{1} \rangle (x \vee y) \vee \langle \bar{2} \rangle y) \\
F(ancestor)_c & = & \langle 1 \rangle \mu x.c \vee \langle 1 \rangle x \vee \langle 2 \rangle x \\
F(ancself)_c & = & \mu x.c \vee \langle 1 \rangle \mu y.x \vee \langle 2 \rangle y \\
F(following)_c & = & F(descself)_{F(fsibling)_{F(ancself)_c}} \\
F(preceding)_c & = & F(descself)_{F(psibling)_{F(ancself)_c}}
\end{array}
$$

As an example of translation, notice that the formula ϕ_2 introduced in Example 4.1 is the translation of the XPath expression e_1 introduced in Example 5.1. More formally $F(e_1)_\top = \phi_2$.

Now, we state that the general translation is trivially correct.

THEOREM 5.2 (XPATH TRANSLATION). *Given a tree structure S, a variable interpretation V, a XPath expression e, and a formula c, we have that*

$$[\![F(e)_c]\!]^S_V = [\![e]\!]^S_{[\![c]\!]^S_V}.$$

The logic also allows capturing regular tree languages which subsume most of schema definitions used in practice (DTDs, XML Schemas, Relax NGs) [20]. The detailed translation of regular tree types into the logic can be found in [11].

6. SATISFIABILITY ALGORITHM

A bottom-up tableau-based algorithm for checking satisfiability of the logic is presented in this section. First, we introduce some definitions: they are mostly shared with [11], but they are extended here with counting features. Then we present the new algorithm capable of handling counting constraints. Finally, we proceed to prove its correctness and to explore its complexity.

6.1 Preliminaries

Consider the least binary relation R_e among formulas, satisfying:

$$
\begin{array}{lcl}
(\phi_1 \wedge \phi_2, \phi_i) & \in & R_e \\
(\phi_1 \vee \phi_2, \phi_i) & \in & R_e \\
(\langle m \rangle \phi, \phi) & \in & R_e \\
(\phi \leq n, \phi) & \in & R_e \\
(\phi > n, \phi) & \in & R_e \\
(\mu x.\phi, exp(\mu x.\phi)) & \in & R_e
\end{array}
$$

The *Fisher-Ladner closure* of a formula ϕ, written $cl(\phi)$, is the set of all subformulas of ϕ where the fixpoint formulas are additionally *unwound* once, formally it is defined as

$$\bigcap \{M \mid M \subseteq F_\phi \wedge \phi_1 \in M \wedge (\phi_1, \phi_2) \in R_e \Rightarrow \phi_2 \in M\},$$

and its *extended closure* as

$$cl^*(\phi) = cl(\phi) \cup \{\neg\psi \mid \psi \in cl(\phi)\}.$$

The set P_ϕ is the set of all propositions used in ϕ along with another proposition, written σ_x, that does not occur in ϕ. Notice the special proposition σ_x allows to model an infinite alphabet by representing all propositions but the the ones occurring in ϕ.

Each formula in the extended closure of a formula ϕ can be seen as a boolean combination of formulas of a set called the *Lean* of ϕ, written $Lean(\phi)$, and defined as

$$\{\langle m \rangle \top \mid m \in \{1, 2, \bar{1}, \bar{2}\}\} \cup \{\langle m \rangle \psi \mid \langle m \rangle \psi \in cl(\phi)\} \cup P_\phi$$
$$\cup \{\psi \leq n \mid \psi \leq n \in cl(\phi)\} \cup \{\psi > n \mid \psi > n \in cl(\phi)\}.$$

For XPath decision problems that involve several XPath expressions (like containment), there is a need to refer several times to the context node from which the XPath expressions apply. This need led us to distinguish such a context node by marking it with another special proposition, named ⓢ. What makes ⓢ special is that it occurs at most once in the tree, and when occurring it can hold at the same node where other proposition holds.

A *type* of a formula ϕ, written t_ϕ, is a non-empty subset of $Lean(\phi)$ such that:

- for each $\langle m \rangle \psi \in Lean(\phi)$, we have that $\langle m \rangle \top \in t_\phi$ when $\langle m \rangle \psi \in t_\phi$;
- $\langle \bar{1} \rangle \top \notin t_\phi$ or $\langle \bar{2} \rangle \top \notin t_\phi$; and
- exactly one atomic proposition, besides ⓢ, occurs in t_ϕ.

The set of types of a formula ϕ is denoted T_ϕ. Types are the logical characterizations of the nodes of a tree.

Given a type t_ϕ, we inductively define the relation $\dot{\in}$ as follows:

- $\top \dot{\in} t_\phi$;
- if $\psi \in Lean(\phi)$ and $\psi \in t_\phi$, then $\psi \dot{\in} t_\phi$;

- if $\phi_1 \dot{\in} t_\phi$ and/or $\phi_2 \dot{\in} t_\phi$ then $\phi_1 \wedge \phi_2 \dot{\in} t_\phi / \phi_1 \wedge \phi_2 \dot{\in} t_\phi$;
- if $exp(\mu x.\psi) \dot{\in} t_\phi$ then $\mu x.\psi \dot{\in} t_\phi$; and
- $\neg\psi \dot{\in} t_\phi$, provided that provided that $\psi \dot{\notin} t_\phi$; where the relation $\dot{\notin}$ is defined in the obvious manner.

Intuitively $\phi \dot{\in} t$ means that the formula ϕ holds at the node represented by t.

In order to represent the transition relation between two nodes through the modalities, we define a compatibility relation between two types. Two types t_ϕ and t'_ϕ are *m–compatible*, written $\Delta_m(t_\phi, t'_\phi)$, iff for every $\langle m\rangle\psi$ and $\langle \bar{m}\rangle\psi$ in $Lean(\phi)$, respectively, we have that:

$$\begin{array}{lcll} \langle m\rangle\psi \in t_\phi & \text{iff} & \psi \dot{\in} t'_\phi, & \text{and} \\ \langle \bar{m}\rangle\psi \in t'_\phi & \text{iff} & \psi \dot{\in} t_\phi. & \end{array}$$

Counting extensions. Cardinality constraints in formulas involve an intrinsinc form of counting in trees. In order to perform such counting, we define the predicates in charge of such task. Given a formula ϕ, we say a set of types $T \subseteq T_\phi$, *satisfies the upper bound cardinality constraints*, written $\#^{\leq}(T)$, if for every formula $\psi \leq n \in Lean(\phi)$ we have that

$$\forall t \in T, \psi \leq n \in t \text{ iff } |\{t' \mid \psi \dot{\in} t', t' \in T\}| \leq n.$$

In the same context, we define the set of types $T \subseteq T_\phi$ satisfying the *lower bound cardinality constraints*, written $\#^{>}(T)$, if for every formula $\psi > n \in Lean(\phi)$ we have that

$$\forall t \in T, \psi > n \in t \text{ if } |\{t' \mid \psi \dot{\in} t', t' \in T\}| > n.$$

The set of formulas occurring in the types of a set of types S^t is written $F(S^t)$.

We will represent binary tree structures as triples (t, T_1, T_2), where t represents the root and T_1 and T_2 are the subtrees linked to the root by the modalities 1 and 2, respectively. Formally, a *types tree* is inductively defined as:

- the empty set; or
- a tuple (t, T_1, T_2), where T_1 and T_2 are types trees, and t is a type.

The mapping $head : \mathcal{T} \mapsto \mathcal{T}$, where $\mathcal{T}$ is a set of types trees, is defined:

- $head(\emptyset) = \emptyset$, and
- $head((t, T_1, T_2)) = t$.

A mapping *type*, from a set of types trees to a set of types is defined as:

- $type(\emptyset) = \emptyset$, and
- $type((t, T_1, T_2)) = \{t\} \cup type(T_1) \cup type(T_2)$.

Given a types tree $T = (t, T_1, T_2)$ we define

- $ch^0(T) = sb^0(T) = T$,
- $ch^1(T) = T_1$, $sb^1(T) = T_2$,
- $ch^i(T) = ch^1(ch^{i-1}(T))$, and $sb^i(T) = sb^1(sb^{i-1}(T))$ $(i \geq 2)$.

We will write $ch(T)$ and $sb(T)$ to denote $ch^1(T)$ and $sb^1(T)$, resp.

6.2 The Algorithm

The algorithm works on a set of types trees. It proceeds in a bottom-up manner, such that new types trees are added until a satisfying model is found, or until no types tree can be added. At each iteration, deeper trees with pending backward modalities (to be fulfilled at later iterations) are built. Also, upper cardinality constraints (formulas in $Lean(\phi)$ with the form $\psi \leq k$) are checked at each iteration, restricting the height of the trees. After each iteration, the types trees are traversed and the lower cardinality constraints (formulas in $Lean(\phi)$ with the form $\psi > k$) are checked in order to verify the minimal required height is fulfilled by a satisfying model. If no more triples can be added and there is no satisfying model, then the formula is unsatisfiable.

We now introduce the algorithm more formally. Given a formula ϕ and a set of types trees $\mathcal{T}$, we define the *update function*, written $Upd(\phi, \mathcal{T})$, to be the set with triples (t, T_1, T_2) such that for $m = 1, 2$:

- $t \in T_\phi$; $T_m \in \mathcal{T}$;
- if $\langle m\rangle\top \in t$ then
  - $\langle \bar{m}\rangle\top \in head(T_m)$,
  - $\Delta_m(t, head(T_m))$, and
  - $\#^{\leq}(\{t\} \cup type(T_m))$ (satisfaction of the upper bound cardinality constraints);
- if Ⓢ $\in t$ then Ⓢ $\notin F(type(T_m))$;
- if Ⓢ $\in type(T_i)$ then Ⓢ $\notin t \cup F(type(T_j))$ $(i, j \in \{1, 2\}$ and $i \neq j)$.

We define the boolean *checking function*, written $Check(\phi, \mathcal{T})$, to be true when there is a types tree $T \in \mathcal{T}$ such that:

- if Ⓢ $\in P_\phi$, then Ⓢ $\in type(T)$;
- $\langle m\rangle\top \notin head(T)$ $(m \in \{\bar{1}, \bar{2}\})$;
- $\#^{>}(type(T))$ (satisfaction of the lower bound cardinality constraints);
- $\phi \dot{\in} t'$ for some $t' \in type(T)$.

Now, consider $X^0 = Upd(\phi, \emptyset)$, $X^{i+1} = Upd(\phi, X^i)$, and

$$sat(\phi) = \begin{cases} 1 & \text{if } Check(\phi, X^k) \text{ and} \\ & \text{there is no } k' \leq k \text{ s.t. } Check(\phi, X^{k'}) \\ 0 & \text{if } X^k = X^{k+1} \text{ and} \\ & \text{there is no } k' \leq k \text{ s.t. } X^{k'} = X^{k'+1}. \end{cases}$$

THEOREM 6.1 (SATISFIABILITY). *A formula ϕ is satisfiable iff $sat(\phi) = 1$.*

6.3 Correctness and Complexity

Notice that the algorithm terminates since the *update* function is clearly monotonic and the set of types is finite. We now show that the algorithm is sound and complete.

Given a types tree T, we say its *equivalent tree* is the tree structure $S = (N, R, L)$, such that:

- $N = type(T)$;
- $R(head(ch^i(T)), 1) = head(ch^{i+1}(T))$ iff $ch^{i+1}(T) \neq \emptyset$;
  $R(head(sb^i(T)), 2) = head(sb^{i+1}(T))$ iff $sb^{i+1}(T) \neq \emptyset$ $(i = 0, 1, 2, \ldots)$; and
- for every $p \in P_\phi$, $L(t) = n$ iff $p \dot{\in} t$.

LEMMA 6.2 (SOUNDNESS). *Given a formula ϕ, if $sat(\phi) = 1$ then ϕ is satisfiable.*

PROOF. We proceed by induction on the structure of ϕ.

The cases when ϕ is either a proposition or a negated proposition are trivial.

If ϕ has the form $\phi_1 \circ \phi_2$ $(\circ \in \{\wedge, \vee\})$, then we know that if $Check(\phi, X^k)$, then $Check(\phi_1, X^k)$ and/or (resp.) $Check(\phi_2, X^k)$, then by induction $[\![\phi_1]\!]_V^S \neq \emptyset$ and/or (resp.) $[\![\phi_2]\!]_V^S \neq \emptyset$ where $S = (N, R, L)$ is the equivalent tree of X^k. Now, we know there is type $t \in type(X^k)$ s.t. $\psi \dot{\in} t$, hence $\phi_1 \dot{\in} t$ and/or $\phi_2 \dot{\in} t$. By the definition of S, we have a node $n \in N$ s.t. $n \in [\![\phi_1]\!]_V^S$ and/or $n \in [\![\phi_2]\!]_V^S$, then $n \in [\![\phi]\!]_V^S$.

When ϕ is $\langle m \rangle \psi$, if $Check(\langle m \rangle \psi, X^k)$, then $Check(\psi, X^k)$ and by induction $[\![\psi]\!]_V^S \neq \emptyset$, where $S = (N, R, L)$ is the equivalent tree of X^k. We know there are two types $t, t' \in type(X^k)$ s.t. $\langle m \rangle \psi \dot{\in} t$, $\psi \dot{\in} t'$ and $\Delta_m(t, t')$, then, by the definition of S, we have two nodes $n, n' \in N$ s.t. $n \in [\![\psi]\!]_V^S$ and $R(n, m) = n'$, hence $n' \in [\![\langle m \rangle \psi]\!]_V^S$.

If ϕ has the form $\psi > n$ and $Check(\psi > n, X^k)$, then $Check(\psi, X^k)$. By induction $[\![\psi]\!]_V^S \neq \emptyset$ where S is the equivalent tree of X^k. By definition of S, we have $|[\![\psi]\!]_V^S| > n$ since $\#^>(X^k)$ holds, therefore $[\![\psi > n]\!]_V^S \neq \emptyset$.

The remaining cases are straightforward. □

LEMMA 6.3 (COMPLETENESS). *If a formula ϕ is satisfiable, then $sat(\phi) = 1$.*

In order to prove completeness we will define a types tree T equivalent to the tree structure satisfying ϕ, i.e., such types tree makes $Check(\phi, \{T\})$ hold. Then we will show there is set X^k produced by Upd s.t. $T \in X^k$.

First, we need some technical machinery.

Given a finite binary tree structure $S = (N, R, L)$ and formula ϕ, we say its *equivalent types tree* is T, such that:

- a type t_n is defined by a node n when $t_n = \{\psi \mid n \in [\![\psi]\!]_V^S \text{ and } \psi \in Lean(\phi)\}$;
- $types(T) = \{t_n \mid \forall n \in N \text{ and } t_n \text{ is defined by } n\}$;
- $head(T) = t_r$, where r is the root of S;
- $head(ch^i(T)) = t_n$, s.t. $head(ch^{i-1}(T)) = t_m$ and $R(n, 1) = m$; and
- $head(sb^i(T)) = t_n$, s.t. $head(sb^{i-1}(T)) = t_m$ and $R(n, 2) = m$ $(i = 1, 2, \ldots)$.

LEMMA 6.4. *Given a formula ϕ, if there is a structure S, a variable interpretation V, and a node n in S, s.t. $n \in [\![\phi]\!]_V^S$, then $check(\phi, \{T\})$ holds and $\phi \dot{\in} t_n$, where T is the equivalent types tree of S w.r.t. ϕ and $t_n \in type(T)$ is defined by n.*

PROOF. Since we are considering only cycle-free formulas, there is an equivalent unfolding of ϕ. Therefore, there is a finite structure satisfying ϕ, say S. We now proceed by structural induction on ϕ.

The base cases where ϕ is either a proposition or a negated proposition are trivial.

If ϕ has the form $\phi_1 \wedge \phi_2$ and $n \in [\![\phi_1 \wedge \phi_2]\!]_V^S \neq \emptyset$, then $n \in [\![\phi_1]\!]_V^S$ and $n \in [\![\phi_2]\!]_V^S$. By induction we know $Check(\phi_1, \{T_1\})$ and $Check(\phi_2, \{T_2\})$ hold, where T_1 and T_2 are the equivalent types trees of S w.r.t. ϕ_1 and ϕ_2, respectively. Moreover, there are two types $t'_n \in type(T_1)$ and $t''_n \in type(T_2)$ s.t. $\phi_1 \dot{\in} t'_n$ and $\phi_2 \dot{\in} t''_n$. Now, consider T to be the equivalent types tree of S w.r.t. $\phi_1 \wedge \phi_2$. Notice T only differs from T_1 and T_2 (resp.) because $Lean(\phi_1 \wedge \phi_2) = Lean(\phi_1) \cup Lean(\phi_2)$, hence it is not hard to see that $Check(\phi_1, \{T\})$ and $Check(\phi_2, \{T\})$ also hold, and $\phi_1, \phi_2 \dot{\in} t_n \in types(T)$ s.t. $t_n = t'_n \cup t''_n$, then $\phi_1 \wedge \phi_2 \dot{\in} t_n$ and so $Check(\phi_1 \wedge \phi_2, \{T\})$.

Consider ϕ is $\langle m \rangle \psi$ and $n_1 \in [\![\langle m \rangle \psi]\!]_V^S$. Hence, there is a node n_2 in S s.t. $n_2 \in [\![\psi]\!]_V^S$ and $R(n_1, m) = n_2$. By induction hypothesis we have that $Check(\psi, \{T'\})$ hold and $\psi \dot{\in} t_{n_2} \in T'$, where T' is the equivalent types tree of S w.r.t ψ. If T is the equivalent types tree of S w.r.t. $\langle m \rangle \psi$, notice it only differs from T' because $t_{n_1} = t'_{n_1} \cup \{\langle m \rangle \psi\}$, where $t_{n_1} \in type(T)$ and $t'_{n_1} \in type(T')$. Then, clearly $Check(\psi, \{T\})$, now, from definition of T, we also have that $\langle m \rangle \psi \dot{\in} t_{n_1}$ and then $Check(\langle m \rangle \psi, \{T\})$ holds.

When ϕ has the form $\psi > k$ and $n \in [\![\psi > k]\!]_V^S$, then clearly there is another node $n' \in [\![\psi]\!]_V^S$. Then by induction hypothesis we have that $Check(\psi, \{T'\})$ and $\psi \dot{\in} t_{n'}$, where T' is the equivalent types tree of S w.r.t. ψ and $t_{n'} \in type(T')$

is defined by n'. If T is the equivalent types tree w.r.t $\psi > k$, then notice it only differs from T' because $t_n = t'_n \cup \{\psi > k\}$, where $t_n \in type(T)$ is the type defined by n. Then, it is easy to see that $\psi > k \dot{\in} t_n$ and $Check(\psi > k, \{T\})$ also hold.

The remaining cases are straightforward. $\square$

Consider a tree structure $S = (N, R, L)$. We define a *subtree* $S' = (N', R', L')$ of S, rooted at node n^r as follow:

- $N' = \{n \mid R(n^r, m) = n, m \in \{1, 2\}$ or $R(n', m') = n, n' \in N', m' \in \{1, 2, \bar{1}, \bar{2}\}\}$;
- $R'(n_1, m) = n_2$ iff $R(n_1, m) = n_2$ and $n_1, n_2 \in N'$; and
- $L'(n) = p$ iff $L(n) = p$ and $n \in N'$.

Consider the subtrees S_1 and S_2 of a tree $S = (N, R, S)$, rooted at r_1 and r_2, resp., s.t. $R(r, 1) = r_1$, $R(r, 2) = r_2$ and r is the root of S. The *height* of S, written $height(S)$, is defined $height(S) = max(height(S_1), height(S_2)) + 1$. If a subtree does not exist, then its height is zero.

LEMMA 6.5. *If a formula ϕ is satisfiable by a structure S, then there is a sequence of sets*

$$X^0 = \emptyset, X^1 = Upd(\phi, X^0), \ldots, X^k = Upd(\phi, X^{k-1}),$$

such that $T \in X^k$, where T is the equivalent types tree of S w.r.t. ϕ.

PROOF. We will proceed by induction on the height of $S = (N, R, L)$.

The base case is easy.

Consider the height of S is k. Let's name S_1 and S_2 the subtrees of S rooted at r_1 and r_2, respectively, such that $R(r, 1) = r_1$ and $R(r, 2) = r_2$, where r is the root in S. Consider the heights of S_1 and S_2 are k_1 and k_2, resp., also notice either $k_1 = k - 1$ or $k_2 = k - 1$. By induction hypothesis we know there are two sequences

$$X_1^1 = Upd(\phi_1, \emptyset), \ldots, X_1^{k_1} = Upd(\phi_1, X_1^{k_1-1}), \text{ and}$$
$$X_2^1 = Upd(\phi_2, \emptyset), \ldots, X_2^{k_2} = Upd(\phi_2, X_2^{k_2-1}),$$

s.t. $T'_1 \in X_1^{k_1}$ and $T'_2 \in X_2^{k_2}$, where ϕ_1 and ϕ_2 are subformulas of ϕ, and T'_1 and T'_2 are the equivalent types trees of S_1 and S_2, resp., w.r.t. ϕ_1 and ϕ_2, resp. Since $Lean(\phi) = Lean(\phi_1) \cup Lean(\phi_2)$, it is not hard to see that there is also a sequence

$$X^1 = Upd(\phi, \emptyset), \ldots, X^{k-1} = Upd(\phi, X^{k-2}),$$

s.t. $T_1, T_2 \in X^{k-1}$, where T_1 and T_2 are the equivalent types tree of S_1 and S_2, resp., w.r.t. ϕ. Then, it is clear there is a $T \in X^k$, s.t. $X^k = Upd(\phi, X^{k-1})$ and T is the equivalent types tree of S w.r.t ϕ. $\square$

Now, from Lemmas 6.4 and 6.5, we conclude completeness.

LEMMA 6.6 (COMPLEXITY). *Given a formula ϕ, the satisfiability problem $[\![\phi]\!]_V^S \neq \emptyset$ is decidable in time $2^{O(n)}$, where $n = |Lean(\phi)|$.*

PROOF. Consider the set X^k after testing $sat(\phi)$. Notice $k \leq 2^n$ and $|X^k| \leq 2^n$, since $|T_\phi| = 2^n$. Now, let's see what happens at each X^i ($i = 1, \ldots, k$). The update function adds triples (t, T_1, T_2), then three traversals are needed, one for each member of the tuples. The first traversal is on T_ϕ and the other two are on X^i, whose sizes do not exceed 2^n. The main tests applied at this stage are the compatibility relation and the upper bound cardinality constraints. The first one is applied on two types, and the second one is on $\{t\} \cup T_j$. The main task of both tests is to check a $\dot{\in}$ relation. It is not hard to see that such relation implies a traversal on a space no greater than n. Hence, the cost to compute each X^i is $2^{O(n)}$. Finally, it is also clear that the time complexity of the *Check* function is $2^{O(n)}$ since $k \leq 2^n$. $\square$

7. CONCLUSION

We presented a sound and complete desicion procedure for a sub-logic of the alternation-free μ–calculus with converse, extended with a counting operator in order to reason on numerical constraints in finite tree structures. The addition of the counting operator does not increase the complexity of the decision procedure, which remains $2^{O(n)}$ in the length n of a formula.

Translations of XPath with cardinality constraints into the logic were introduced. This yields a characterization of an expressive fragment of XPath, for which static analysis methods were not known so far.

The XPath fragment we consider in this paper performs global counting in trees. We are currently refining our approach in order to consider local counting as in $\rho_1[count(\rho_2) \leq n]$. Also, we are implementing the decision procedure.

Among the further research directions, we are interested in the developing of a generalization of our method, in order to reason efficiently on more sofisticated counting approaches in both, trees and graphs.

8. REFERENCES

[1] L. Afanasiev, P. Blackburn, I. Dimitriou, B. Gaiffe, E. Goris, M. Marx, and M. de Rijke. PDL for ordered trees. *Journal of Applied Non-Classical Logics*, 15(2):115–135, 2005.

[2] P. Barceló and L. Libkin. Temporal logics over unranked trees. In *LICS*, pages 31–40. IEEE Computer Society, 2005.

[3] M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. *J. ACM*, 55(2):1–79, 2008.

[4] P. Blackburn, J. F. A. K. van Benthem, and F. Wolter. *Handbook of Modal Logic, Volume 3 (Studies in Logic and Practical Reasoning)*. Elsevier Science Inc., New York, NY, USA, 2006.

[5] J. Clark and S. DeRose. XML path language (XPath) version 1.0. W3C recommendation, November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

[6] S. Dal-Zilio, D. Lugiez, and C. Meyssonnier. A logic you can count on. *SIGPLAN Not.*, 39(1):135–146, 2004.

[7] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *FOCS*, pages 368–377. IEEE, 1991.

[8] D. C. Fallside and P. Walmsley. XML Schema part 0: Primer second edition. W3C recommendation, October 2004. `http://www.w3.org/TR/xmlschema-0`.

[9] W. Fan, C.-Y. Chan, and M. Garofalakis. Secure XML querying with security views. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 587–598, New York, NY, USA, 2004. ACM.

[10] K. Fine. In so many possible worlds. *Notre Dame J. of Formal Logic*, 13(4):516–520, 1972.

[11] P. Genevès, N. Layaïda, and A. Schmitt. Efficient static analysis of XML paths and types. In *PLDI '07: Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 342–351, New York, NY, USA, 2007. ACM Press.

[12] P. Genevès and J.-Y. Vion-Dury. Logic-based XPath optimization. In *DocEng '04: Proceedings of the 2004 ACM symposium on Document engineering*, pages 211–219, New York, NY, USA, 2004. ACM Press.

[13] B. Hollunder and F. Baader. Qualifying number restrictions in concept languages. In *KR*, pages 335–346, 1991.

[14] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer, 1999.

[15] M. Jurdzinski and R. Lazic. Alternation-free modal mu-calculus for data trees. In *LICS*, pages 131–140. IEEE Computer Society, 2007.

[16] F. Klaedtke and H. Rueß. Monadic second-order logics with cardinalities. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 681–696. Springer, 2003.

[17] D. Kozen. Results on the propositional μ-Calculus. In M. Nielsen and E. M. Schmidt, editors, *ICALP*, volume 140 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 1982.

[18] O. Kupferman, U. Sattler, and M. Y. Vardi. The complexity of the graded μ-calculus. In A. Voronkov, editor, *CADE*, volume 2392 of *Lecture Notes in Computer Science*, pages 423–437. Springer, 2002.

[19] A. Møller, M. Ø. Olesen, and M. I. Schwartzbach. Static validation of XSL Transformations. *ACM Transactions on Programming Languages and Systems*, 29(4), July 2007.

[20] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Trans. Internet Techn.*, 5(4):660–704, 2005.

[21] H. Seidl, T. Schwentick, and A. Muscholl. Numerical document queries. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 155–166, New York, NY, USA, 2003. ACM.

[22] H. Seidl, T. Schwentick, A. Muscholl, and P. Habermehl. Counting in trees for free. In J. Díaz, J. Karhumäki, A. Lepistö, and D. Sannella, editors, *ICALP*, volume 3142 of *Lecture Notes in Computer Science*, pages 1136–1149. Springer, 2004.

[23] Y. Tanabe, K. Takahashi, M. Yamamoto, A. Tozawa, and M. Hagiya. A decision procedure for the alternation-free two-way modal mu-calculus. In B. Beckert, editor, *TABLEAUX*, volume 3702 of *Lecture Notes in Computer Science*, pages 277–291. Springer, 2005.

[24] M. Y. Vardi. Why is modal logic so robustly decidable? In N. Immerman and P. G. Kolaitis, editors, *Descriptive Complexity and Finite Models*, volume 31 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 149–184. American Mathematical Society, 1996.

[25] M. Y. Vardi. Reasoning about the past with two-way automata. In *ICALP '98: Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, pages 628–641, London, UK, 1998. Springer-Verlag.

[26] M. Y. Vardi and L. J. Stockmeyer. Improved upper and lower bounds for modal logics of programs: Preliminary report. In *STOC*, pages 240–251. ACM, 1985.